8

Handling Exceptions

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- Define PL/SQL exceptions
- Recognize unhandled exceptions
- List and use different types of PL/SQL exception handlers
- Trap unanticipated errors
- Describe the effect of exception propagation in nested blocks
- Customize PL/SQL exception messages

Example of an Exception

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :'
                        ||v_lname);
END;
```

Example of an Exception

```
DECLARE
  v_lname VARCHAR2(15);
BEGIN
  SELECT last_name INTO v_lname
  FROM employees
  WHERE first_name='John';
  DBMS_OUTPUT.PUT_LINE ('John''s last name is :'
                         ||v_lname);
EXCEPTION
  WHEN TOO_MANY_ROWS THEN
  DBMS_OUTPUT.PUT_LINE (' Your select statement
  retrieved multiple rows. Consider using a
  cursor.');
END;
/
```

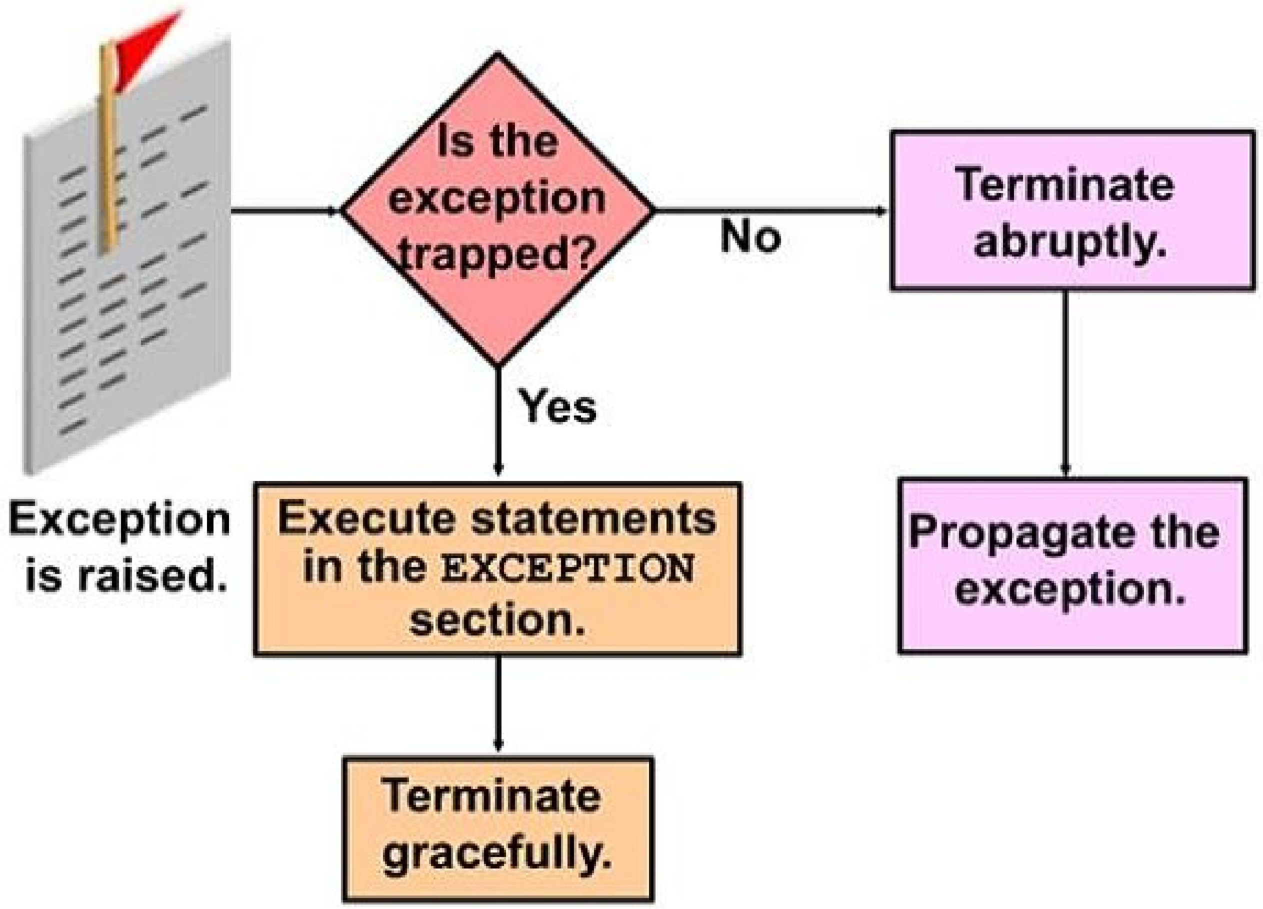
```
anonymous block completed
 Your select statement retrieved multiple
  rows. Consider using a cursor.
```

Handling Exceptions with PL/SQL

- An exception is a PL/SQL error that is raised during program execution.
- An exception can be raised:
  - Implicitly by the Oracle server
  - Explicitly by the program
- An exception can be handled:
  - By trapping it with a handler
  - By propagating it to the calling environment

ORACLE

Handling Exceptions



Exception Types

- Predefined Oracle server
- Non-predefined Oracle server

$\left.\right\}$ **Implicitly raised**

- User-defined

Explicitly raised

ORACLE

Trapping Exceptions

Syntax:

```
EXCEPTION
  WHEN exception1 [OR exception2 . . .] THEN
    statement1;
    statement2;
    . . .
  [WHEN exception3 [OR exception4 . . .] THEN
    statement1;
    statement2;
    . . .]
  [WHEN OTHERS THEN
    statement1;
    statement2;
    . . .]
```

Guidelines for Trapping Exceptions

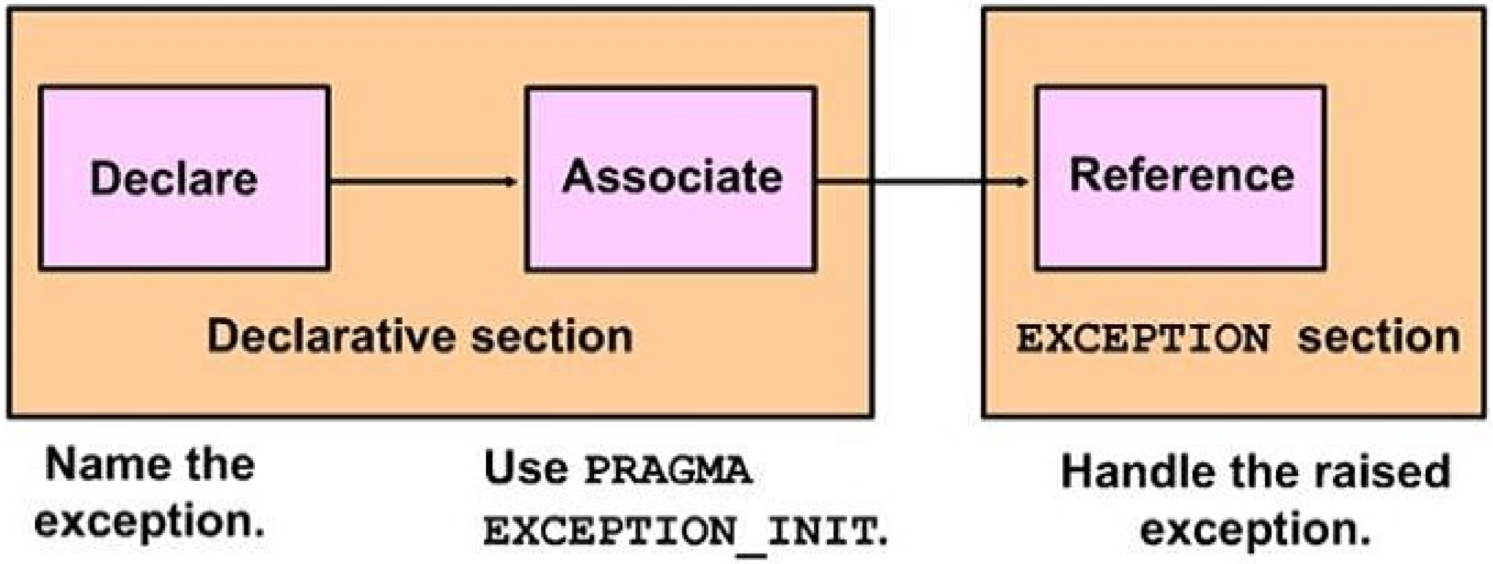
- The `EXCEPTION` keyword starts the exception-handling section.
- Several exception handlers are allowed.
- Only one handler is processed before leaving the block.
- `WHEN OTHERS` is the last clause.

ORACLE

Trapping Predefined Oracle Server Errors

- Reference the predefined name in the exception-handling routine.
- Sample predefined exceptions:
  - `NO_DATA_FOUND`
  - `TOO_MANY_ROWS`
  - `INVALID_CURSOR`
  - `ZERO_DIVIDE`
  - `DUP_VAL_ON_INDEX`

Trapping Non-Predefined Oracle Server Errors



Non-Predefined Error

To trap Oracle server error number `-01400` ("cannot insert `NULL`"):

```
DECLARE
 e_insert_excep EXCEPTION;                                  (1)
 PRAGMA EXCEPTION_INIT(e_insert_excep, -01400);             (2)
BEGIN
 INSERT INTO departments
 (department_id, department_name) VALUES (280, NULL);
EXCEPTION                                                   (3)
 WHEN e_insert_excep THEN
   DBMS_OUTPUT.PUT_LINE('INSERT OPERATION FAILED');
   DBMS_OUTPUT.PUT_LINE(SQLERRM);
END;
/
```

```
anonymous block completed
INSERT OPERATION FAILED
ORA-01400: cannot insert NULL into ("ORA41"."DEPARTMENTS"."DEPARTMENT_NAME")
```

Functions for Trapping Exceptions

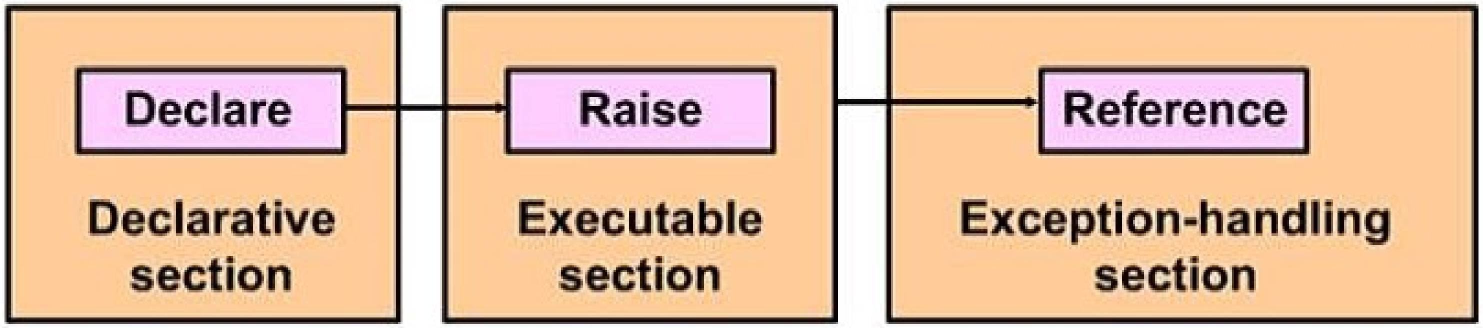
- `SQLCODE`: Returns the numeric value for the error code
- `SQLERRM`: Returns the message associated with the error number

Functions for Trapping Exceptions

```
DECLARE
  error_code      NUMBER;
  error_message   VARCHAR2(255);
BEGIN
...
EXCEPTION
...
  WHEN OTHERS THEN
    ROLLBACK;
    error_code := SQLCODE ;
    error_message := SQLERRM ;
   INSERT INTO errors (e_user, e_date, error_code,
   error_message) VALUES(USER,SYSDATE,error_code,
   error_message);
END;
/
```

ORACLE

Trapping User-Defined Exceptions



Name the exception.

Explicitly raise the exception by using the `RAISE` statement.

Handle the raised exception.

Trapping User-Defined Exceptions

```
DECLARE
  v_deptno NUMBER := 500;
  v_name VARCHAR2(20) := 'Testing';
  e_invalid_department EXCEPTION;
BEGIN
  UPDATE departments
  SET department_name = v_name
  WHERE department_id = v_deptno;
  IF SQL % NOTFOUND THEN
    RAISE e_invalid_department;
  END IF;
  COMMIT;
EXCEPTION
WHEN e_invalid_department THEN
  DBMS_OUTPUT.PUT_LINE('No such department id.');
END;
/
```

1

2

3

```
anonymous block completed
No such department id.
```

Propagating Exceptions in a Subblock

Subblocks can handle an exception or pass the exception to the enclosing block.

```
DECLARE
  . . .
  e_no_rows       exception;
  e_integrity     exception;
  PRAGMA EXCEPTION_INIT (e_integrity, -2292);
BEGIN
  FOR c_record IN emp_cursor LOOP
    BEGIN
     SELECT ...
     UPDATE ...
     IF SQL%NOTFOUND THEN
       RAISE e_no_rows;
     END IF;
    END;
  END LOOP;
EXCEPTION
  WHEN e_integrity THEN ...
  WHEN e_no_rows THEN ...
END;
/
```

ORACLE

`RAISE_APPLICATION_ERROR` Procedure

Syntax:

```
raise_application_error (error_number,
             message[, {TRUE | FALSE}]);
```

- You can use this procedure to issue user-defined error messages from stored subprograms.
- You can report errors to your application and avoid returning unhandled exceptions.

`RAISE_APPLICATION_ERROR` Procedure

- Used in two different places:
  - Executable section
  - Exception section
- Returns error conditions to the user in a manner consistent with other Oracle server errors

RAISE_APPLICATION_ERROR Procedure

Executable section:

```
BEGIN
...
  DELETE FROM employees
     WHERE  manager_id = v_mgr;
  IF SQL%NOTFOUND THEN
     RAISE_APPLICATION_ERROR(-20202,
       'This is not a valid manager');
  END IF;
   ...
```

Exception section:

```
...
EXCEPTION
   WHEN NO_DATA_FOUND THEN
     RAISE_APPLICATION_ERROR (-20201,
        'Manager is not a valid employee.');
END;
/
```

ORACLE

Quiz

You can trap any error by including a corresponding handler within the exception-handling section of the PL/SQL block.

1. True
2. False

Summary

In this lesson, you should have learned how to:

- Define PL/SQL exceptions
- Add an `EXCEPTION` section to the PL/SQL block to deal with exceptions at run time
- Handle different types of exceptions:
  - Predefined exceptions
  - Non-predefined exceptions
  - User-defined exceptions
- Propagate exceptions in nested blocks and call applications

ORACLE